UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/650,238	08/27/2003	Wolfram Schulte	3382-65592	6392

26119 7590 04/03/2008

KLARQUIST SPARKMAN LLP
121 S.W. SALMON STREET
SUITE 1600
PORTLAND, OR 97204

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
04/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/650,238	SCHULTE ET AL.	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *28 February 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-33* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-33* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 2/28/08.

As indicated in Applicant's response, claims 1, 19, 31 have been amended. Claims 1-31 are pending in the office action.

Claim Objections

2. Claim 19 is objected to because of the following informalities: the term 'exectuable' (li. 12) appears to be a typographical mistake. Appropriate correction is required.

Claim Rejections - 35 USC § 112

3. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

4. Claims 1-33 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Claims 1, 19, 31 are rejected under 35 U.S.C. 112, second paragraph, as being incomplete for omitting essential structural/functional cooperative relationships of elements, such omission amounting to a gap between the necessary structural connections. See MPEP § 2172.01. The omitted cooperative relationships are: the time-related relationship gap linking the execution recited as 'during execution of ... executable computer program' of the program limitation and the actions of *receiving a reflection*, *producing of data domain*, the use of 'input values' of said data domain, and *targeting testing* and *determining* 'whether … when executing… as input'.

The recited steps recited as 'receiving' (a reflection program), 'producing' (data domain … based the reflection … executable program, 'targeting testing' during execution and

'determining' whether ... program behaves correctly, in a whole entail a sequence in time that dictates the achieving of one step in order for the next step to be effectuated, the latter step using the result of the former step. However, the claim recites only one 'during execution of the program' without discriminating which instances of execution (if any) and which executable code (if any) for one to be apprised about how the relationship between the above steps would make some sense. That is, the claim recites for each of the above steps having result being coined from what appears to be what amounts to a same instance of executing a same executable ('during execution of the computer executable program'). From reading the Specifications, there are disjoint instances where code execution would be for testing and verification based on value ranges (Specifications Fig. 4, pg. 10) but this testing would be subsequent or distinct in time context from the runtime in which to obtain reflection of another instance of executable (Specifications pg. 10, 18). The 'during execution of the program' as repeated in the claim (e.g. claim 1, li. 3, 6, 9, 11 or claim 33, li. 3, 9, 12, 14) seems to be one limitation about one executable runtime (during the execution of the executable program). This single instance of execution would not enable one of ordinary skill to construe that the claimed sequence of getting reflection data, deriving the data domain, and verification test as observed from above is reasonably clarified, even with the support from the Specifications. Absent a definite teaching about a time factor thus amounting to a clear gap in enabling one to construe the crux of the invention, the above lack of structural/functional cooperative relationship fails to teach (for one of ordinary skill in the art) a reasonable understanding; e.g. is there a utility than enables based on some break-and-resume setting, for enabling how the result from each of many steps/stages to be timely available as input for use between execution stages in the course of just ONE single

execution flow? In short, the claim seems to omit some teaching as to enable how one instance of execution ('during execution of the program') can simultaneously read reflection, derived its data domain, use the range of values therein as input to feed back into execution (of the very program) so to target testing such program to verify the correctness of the data domain; notwithstanding the lack of parallel therefor from the Disclosure, according to which the verification test seems not the same instance of runtime wherein reflection data is obtained. Hence the 'during execution' limitation would not be given full patentable weight, and the 'during execution… ' phrase would be treated as EITHER one instance for obtaining reflection data, OR one instance to verify the data domain inputs, but NOT necessarily instances in (at least) two distinct time contexts using (at least) two version of executables (emphasis added). Appropriate corrective action is strongly advised.

Claims 2-18, 20-32 fail to remedy to the lack of structural and time relationship between execution of program in terms of the same 'during execution' requirement and the sequence of actions recited; hence are also rejected for deficiency in omitting clear teaching as to enable the construction of what inventor deemed his invention.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> A person shall be entitled to a patent unless –
>
> (a) a patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 1-33 are rejected under 35 U.S.C. 103(a) as being unpatentable under Davidson et al., USPN: 6,083,276(hereinafter Davidson).

As per claim 1, Davidson discloses a computer implemented method for producing a data domain for a data structure element of a executable computer program, the method comprising:

receiving domain configuration information corresponding to the data structure element (e.g. Fig. 3A, 3B, 3C);

receiving a reflection of the computer program (e.g. bean a property Table 1; *BeanInfo* - Fig. 5; *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60; *accessor ... methods* … value of a property – col. 25, lines 1-8); and

producing the data domain (e.g. Fig. 4B, 4C – Note: mapping corresponding descriptor or attribute for a method or class reads on data domain combining configuration information with reflection of beans components – see col. 25, line 12 to col 26, line 9) based on the domain configuration information and the program reflection,

the data domain representing a limited set of data values (e.g. col. 26, lines 18-29, 38-48) to be used as input during execution of the computer program (e.g. Execute 436 – Fig. 4C – Note: exception error reads on validating or verifying on a range of value or scope of expected constructs being used as input; and whereby error is generated – as a result of on mismatch of expected parameter or method attributes against domain information **reads on** during execution of program to be tested) for testing execution of the executable computer program;

targeting testing of the computer program to use only values for the data structure element that fall within the data structure (e.g. *expected parameters* – col. 25, lines 8-32;

canonical names ... scope path named ... more attributes ... remain to be mapped – col. 25, line 34 to col. 26, line 9 – Note: verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure – see *propertyDescriptor, parameters ... write method* - Fig. 4C) and determining whether the executable computer program behaves correctly when executing using targeted values falling within the data domain as input (e.g. generate error 458 - Fig 4D – Note: error generated reads on dynamic execution and determining; wherein verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure or domain info of the bean container or Property scope – e.g. <Style type=Link … value= … value=… … value= … </Style> , pg. 21 lines 40-48; col. 26, lines 18-28; or enumerating values in a array – see class[] and Object[] lines 38-48).

But Davidson does not explicitly teach obtaining reflection of an executable program **during its execution**, the program to be used to **target efficient testing of behavior of the executable program during its execution**, the executable computer program **having been compiled into executable form**. But Davidson discloses an execution environment (generate error 458 - Fig 4D – Note: error generated reads on dynamic execution and determining) to dynamically determine whether some expected value or range of Java elements composing a container scope for a specific method or bean container, such runtime determination being based on the domain data provided by the results from parsing XML into a structure (e.g. Fig. 3B), and the type of metadata (e.g. Fig. 4C) being received from using bean info descriptor and underlying

API (e.g. *executable 'accessor' methods ... obtaining and setting the value of a property* 320 – col. 24, line 50 to col. 25, line 7) or as a alternate API to introspect the runtime bean (e.g. col. 26, lines 55-60). Standardardized APIs thus mentioned, being y invoked when executing Sun Microsystems Java or bean code to dynamically obtain additional information about the code strongly entails data in order to help enhance the code or prevent its potential undesired use of runtime resources similar to debug or validation testing. Hence, the concept of testing a target executable program as recognized from above, such that testing would be based on metadata received from a runtime (i.e. during execution of an executable program) introspector API or *standard reflection methods* is strongly evidenced as an alternative. It would have been obvious for one skill in the art at the time the invention was made to implement Sun Microsystems API in testing a target bean code as endeavored by Davidson, so that reflection data is obtained using such runtime API (Davidson: *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60) and to test such target compiled bean based on the metadata constructed based on the introspection method to yield sufficient metadata input to the determination step as set forth in Davidson's executable code testing set forth above.

As per claim 2, see Fig. 1 for computer readable media having computer executable instructions for performing the method of claim 1.

As per claim 3, refer to claim 1 for a listing of data structure elements of the computer program as reflection of computer program (e.g. Fig. 2).

As per claim 4, Davidson discloses annotating code of the computer program (e.g. comment 302 – Figs. 3; e.g. col. 29-36 Appendix A for ADML for comments between special tags <!-- ... -->) with the domain configuration information.

As per claims 5-6, Davidson discloses computer readable media having computer executable instructions for compiling the code of the computer program annotated with the domain configuration information for producing the data domain (Fig. 4B, 4C cols 21-28) according to its domain configuration information.

As per claims 7-8, Davidson discloses the domain configuration information comprising one or more expressions (e.g. BML – col. 8-col. 10 – Note: tag specification *<Foo Att1 = Value 1 Att2=Value2 ... />* reads on explicit denotation of domain to be produced) for explicitly denoting the data domain to be produced corresponding in form to one that is applicable to the data structure element; wherein the expressions comprise methods and functions (e.g. parameter method Fig. 4C; METHOD, ARGUMENTS - col. 16, lines 28-42; Fig. 5; *CALL calls a method ... Attributes* - Appendix A, col. 47, bottom - Note: beans constructs being described in BML language as method and arguments reads on methods and functions) defined within the code of the computer program, which are exposed via the reflection of the computer program.

As per claim 9, Davidson discloses wherein the data structure element is a data type with one or more fields and the form of the explicitly expressed data domain is a set of values of the fields comprising the data type (e.g. TYPE – col. 16, lines 54-63).

As per claim 10, Davidson discloses wherein the data structure element is a method (re claim 9) and the form of the explicitly expressed data domain is a set of tuples of parameters (e.g. *GET FIND CONSTANT|ARRAY| ANY| ALL|NOT* – Appendix A, col. 29-30) of the method.

As per claim 11, Davidson discloses wherein the data structure element is a field or a parameter of a designated type (Table 2, pg. 19; TYPE ID - lines 55-63, col. 16) and the form of the explicitly expressed data domain is an enumeration of values (e.g. <VALUE …</VALUE>

lines 55-63, col. 16; lines 28-42, col. 16) of the designated type corresponding to the field or the parameter.

As per claim 12, Davidson discloses inheriting (e.g. Fig. 3C; *children* - col. 10, lines 50-65; *Child component ...Parent component* – Fig. 4D; lines 9-29 - col. 13) the data domain to be produced from the data domain of other related data structure elements.

As per claim 13, Davidson discloses a data type comprising a plurality of sub-types and a selection of one or more of the plurality of sub-types wherein the data domain to be produced for the data type is a union of data domains of the sub-types (**P** tag … Table 2, col. 19; Style tag…Table 3, col. 20 – Note: paragraph and style tag with subtypes read on plurality of subtype and selection from an union of subtypes) belonging to the selection.

As per claim 14, Davidson discloses data structure element being a field or a parameter of a designated type (Fig. 5; Table 2, pg. 19; **TYPE ID** - lines 55-63, col. 16; cols. 17-18) and the domain configuration information comprises information indicating that the data domain to be produced for the field or the parameter is inherited (e.g. Fig. 3C; *children* - col. 10, lines 50-65; *Child component... Parent component* – Fig. 4D; lines. 9-29 - col. 13)from the data domain of their designated type.

As per claims 15-16, Davidson discloses domain configuration information related to producing the data domain for the data structure element by applying domain generation techniques on other selected data domains; and filtering the result of the applying domain generation technique step using a predicate (steps 410, 418, Fig. 4B; Match 424, Conversion 432, More attributes 440 – Fig. 4C; step 456, Fig. 4D – Note: code to map components as called

by description information with respect to parse algorithm reads on predicates for filtering data from information domain into data domain – see Java pseudo-code col. 26, 28).

As per claims 17-18, Davidson discloses data structure element is a data type with a plurality of fields (e.g. lines 30-34, 55-58 -col. 9; lines 30-43, col. 16; Table 2, col. 19) and the other data domains are data domains of the fields (re claim 1 or Fig. 4B, 4C); wherein the data structure element is a method (Table 1, col. 17-18) and the other data domains are data domains of the parameter of the method (re claim 1; see *write method 504* - col. 25).

As per claim 19, Davidson discloses a system for producing a data domain for a data structure element of a executable computer program, the system comprising a computer apparatus configured to perform actions of a domain configuration manager for

receiving domain configuration information (e.g. 3A, 3B, 3C) corresponding to the data structure element and

using a reflection of the executable computer program (e.g. bean a property Table 1; *BeanInfo* -Fig. 5; *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60; *accessor ... methods …* value of a property – col. 25, lines 1-8) to produce the data domain for the data structure element according to the domain configuration information (Fig. 4B, 4C – Note: mapping corresponding descriptor or attribute for a method or class reads on data domain combining configuration information with reflection of beans components – see col. 25, line 12 to col 26, line 9);

the data domain representing a limited set of data values (e.g. col. 26, lines 18-29, 38-48; *setting the value of a property 320* – col. 24, line 50 to col. 25, line 7) to be used as input for testing execution of the executable computer program (Execute 436 – Fig. 4C – Note: exception

error reads on validating or verifying on a range of value or scope of expected constructs being used as input; and whereby error is generated – as a result of on mismatch of expected parameter or method attributes against domain information **reads on** during execution of program to be tested - e.g. <Style type=Link … value= … value=… … value= … </Style> , pg. 21 lines 40-48; col. 26, lines 38-48; enumerating values in a array – see class[] and Object[] lines 38-48); and

controlling testing of the computer program to use only values for the data structure element that fall within the data structure (e.g. *expected parameters* – col. 25, lines 8-32; *canonical names ... scope path named ... more attributes ... remain to be mapped* – col. 25, line 34 to col. 26, line 9 – Note: verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure – see *propertyDescriptor, parameters ... write method* - Fig. 4C).

But Davidson does not explicitly teach obtaining reflection of an executable program during its execution, the program to be used to target efficient testing of behavior of the executable program during its execution, the executable computer program having been compiled into executable form. However, the above limitation has been addressed as obvious in light of Davidson's alternative of using standard reflection methods in Suns Java runtime as known as *introspector* API invoked within executing a program that has been compiled as target program as endeavored by Davidson's method.

As per claims 20-21, Davidson discloses a graphical user interface communicative with the domain configuration manager for receiving the domain configuration information and transferring (Fig. 1; col. 18-40 -col.28; Error Message – Fig 4B) the domain configuration

information to the domain configuration manager; a GUI for receiving user input related to the domain configuration information (e.g. user and application-generated 'events' lines 15-40 -col. 10).

As per claim 22, Davidson discloses domain configuration manager for reading the reflection of the computer program to identify the data structure element for its domain configuration (e.g. Fig. 3B, 3C; Fig. 4B).

As per claim 23, Davidson discloses wherein the data structure element is a data type and the domain configuration manager is operable for producing the data domain for the data type according to an explicit expression indicative of the data domain of the data type (refer to rationale of claims 13-14).

As per claim 24, Davidson discloses wherein the explicit expression comprises methods and functions defined within the computer program (e.g. col. 17-18) and exposed to the domain configuration manager via the reflection of the computer program (Table 1, col. 17-18).

As per claim 25 Davidson discloses wherein the data structure element is a method and the domain configuration manager is operable for producing the data domain as a set of tuples of parameters of the method according to an explicit expression of the domain configuration information (refer to claim 10).

As per claim 26 Davidson discloses wherein the data structure element is a field or a parameter of a declared type and the data configuration manager is operable for producing the data domain according to an explicit expression whose result is an enumeration of values of the declared type (refer to claim 11).

As per claim 27 Davidson discloses wherein the data structure element is a data type with sub-types and the data configuration manager is operable for producing the data domain for the data type through inheritance as a union of data domains of its selected sub-types (refer to claim 13).

As per claim 28 Davidson discloses wherein the data structure element is a data type and the data configuration manager is operable for producing the data domain for the data type by applying a domain generation technique to one or more fields of the data type (refer to claim 15).

As per claim 29, Davidson discloses wherein the domain generation technique is a Cartesian product (e.g. *mapper 122, 124* – Fig. 1; *map 418* – Fig. 4b; *Match 424*, Fig. 4C; *BeanInfo Mapper* - Fig. 5 - Note: a Cartesian or Cross product between 2 sets A and B is defined as the set of all pairs $\{a, b\}$ such that a is an element of the set A and b is an element of the set B; i.e. mapping an element of A with a corresponding element of B) of the selected fields of the data type and the domain configuration manager is further operable for applying a constraint specified in the domain configuration information to the Cartesian product for producing the data domain for the data type (refer to claim 16 for filtering constraint using predicate).

As per claim 30 Davidson discloses wherein the data structure element is a field or a parameter of a declared type and the domain configuration manager is operable for producing the data domain for the field or the parameter as the data domain of their respective declared type through inheritance (refer to claim 14).

As per claim 31, Davidson discloses wherein the data structure element is a method (re claim 24-25) and the domain configuration manager is operable for producing the data domain

for the method by applying a domain generation technique to the parameters of the method (re claim 15).

As per claim 32, Davidson discloses wherein the domain configuration technique is a Cartesian product (re claim 29) of the data domains of the parameters (re claims 26 and 30) of the method and the data configuration manager is further operable for applying a constraint (refer to claim 16) for filtering constraint using predicate) to the result of the Cartesian product for producing the data domain for the data type.

As per claim 33, Davidson discloses a computer-based system for producing data domains of data structure elements of a computer program, the system comprising a computer apparatus; and means for:

receiving, on the computer apparatus, domain configuration information corresponding to the data structure elements;

reading, on the computer apparatus, a reflection of the executable computer program; and

processing, on the computer apparatus, the domain configuration information and the reflection to produce and output the data domains (e.g. Fig. 1, 3; Fig .5 – Note: attributes and their expected instances within Descriptor scope reads on data domain being **outputted** for verification via mapping) corresponding to the data structure elements; the data domain representing a limited set of data values (e.g. col. 26, lines 18-29, 38-48; *setting the value of a property 32*0 – col. 24, line 50 to col. 25, line 7) to be used as input for testing execution of the executable computer program (Execute 436 – Fig. 4C – Note: exception error reads on validating or verifying on a range of value or scope of expected constructs being used as input; and

whereby error is generated – as a result of on mismatch of expected parameter or method attributes against domain information **reads on** during execution of program to be tested)

for limiting testing during execution of the executable computer program to use only values for the data structure element that fall within the data structure (refer to claim 1);

all of which limitations having been addressed in claim 1.

But Davidson does not explicitly teach obtaining reflection of an executable program during its execution, the program to be used to target efficient testing of behavior of the executable program during its execution, the executable computer program having been compiled into executable form. However, the above limitation has been addressed in claim 1.

Response to Arguments

7. Applicant's arguments filed 2/28/08 have been fully considered but they are either moot or not persuasive. Following are the Examiner's observation in regard thereto.

35 USC § 102 Rejection:

(A) Applicants have submitted that for claim 1, Davidson's actions cannot 'receive a reflection of executable computer program as exemplified in the .RTM portion of the instant Application; and that Java beans or *methods being written* as proffered in the Office Action cannot be executable code (Appl. Rmrks pg. 10, middle; pg. 11, middle). The new grounds of rejection to address this runtime invoking of a method to read or obtain this reflection as bean metadata has been in place; and this would render the argument largely moot.

(B) Applicants have submitted that nowhere in Davidson is there any mention of compilation into executable program such that reflection as required is not seen in Davidson's mapping process to yield no more than incomplete program (Appl. Rmrks pg. 11, bottom, pg. 12 top).

The limitation as to invoke methods for reflection reading during runtime of a executable code as well known in Sun Microsoft standard Java API as suggested in Davidson, has rendered the 'during execution of the executable' limitation obvious. Besides, the 'execution of the executable program' concerned here has been deemed largely indefinite language when compared to the corresponding parts of the Disclosure, which is not enabling one to construe exactly when executable program is implicated in the above argument. As a result of the USC 112 rejection, the 'executable program' is not bearing much of a patentable weight beyond broad interpretation from the Office Action. The Argument about this 'executable program' to read reflection is not persuasive and rather moot in view of the grounds of rejection by virtue of the changes to the scope of this 'reflection' limitation.

(C) Applicants have submitted that Davidson does not teach 'determining whether the executable program … falling within the data domain as input' based on Davidson's mappings being made well prior to creation of the executable (Appl. Rmrks pg. 12 middle). The language regarding the accuracy of this 'executable program' is being highly questioned as not having been describe in the claim as being specific to one instance in time, nor does this executable nature get proper support from the Specifications in order for this limitation to yield a meaningful structural relationship between all the steps being claimed. In this instance, as demonstrated by the cited portions of Davidson, 'execution' would be treated as a runtime by which code under execution yield error; or code for effectuating calls to *accessor* methods to retrieve bean information or class properties, absent a definite teaching (e.g. including a clear time-based relationship between execution instance of one program with respect to each step claimed) in the claims as set forth in the USC 112, rejection. Thus, the argument about code not

yet created is deemed not sufficient. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

(D) Applicants have submitted that 'data domain' as provided in the Office Action does not support 'a limited set of data values to be used for input for testing' (Appl. Rmrks, pg. 12, bottom). This limitation as newly added has been interpreted as range or scope of declaration, variable, property (or value thereof – see Davidson: *setting the value of a property 32*0 – col. 24, line 50 to col. 25, line 7) for which a special bean container or class would have to validate in order to effectuate proper data value conformance. The argument would be deemed moot because the above (limited set of data values) limitation is a amendment, i.e. not commensurate with last Office Action, that would not qualify for an Examiner's rebut. It is also noted that there is a effect of expressing a range of data destined to be validated; or a scope of values enumerated in some list object or array to check in the fact that Davidson discloses a scope for a type=Link in which a set of values is depicted (see col 21, <Style> … </Style>) and a Property object or aParameters[] for which for each attribute a attribute value is also imparted or written as one element belonging to that scope or set (see col. 26, lines 18-29, 38-44)

(E) Applicants have submitted that based on the Advisory Action remarks, it is contended that Davidson does not teach testing of an executable computer program (Appl. Rmrks pg. 13, middle). This appears to be a observation that is not applying the proper compliance respective to a CFR 1.111b type of rebut to a rejection. It is not revealed from the above, that Davidson fails to disclose – in reference to specific cited portions in Davidson, how a specific construct of

the claim language is not anticipated because of some factual teaching to the contrary. Regarding how Davidson does not anticipate, the current rejection has been based on a 103(a) type of rationale, as opposed to a USC § 102 anticipation.

(F) As per claims 19, 33, the arguments are implied here (Appl. Rmrks pg. 13 bottom to pg. 14) to be same as those destined for rebutting the rejection of claim 1. These claims will stand rejected by virtue of the Examiner's observations made in sections A-E above.

In all, the claims 1-33 will stand rejected as set forth in the Office Action.

Conclusion

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished

applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

March 31, 2008